Unityはじめるよ

～初心者向け講座！簡易レースゲーム制作でUnityを覚えよう～

統合開発環境を内蔵したゲームエンジン
http://japan.unity3d.com/

※いろんな職業の方が見る資料なので説明を簡単にしてある部分があります。正確には本来の意味と違いますが上記理由のためです。ご了承ください。
この資料内の一部の画像、一部の文章はUnity公式サイトから引用しています。

資料の内容

- レースゲームを作るにあたって考えること
- 設計
- 実装

まえおき

この資料で伝えたいことは、
Unityの使い方や、ゲームを作る上での
考え方をまとめたものです。

企画やゲームルールは深く追求した内容
ではありません。

また、初心者向けではありますが、
フォルダやゲームオブジェクトを作成できる
くらいの基本操作はできることが前提の
内容となります。

レースゲームを作るにあたって考えること

ルールを明確にし、仕様を決める

ルールは以下のとおり。

・コースを３周したらゴール
・敵(AI)と順位を競う

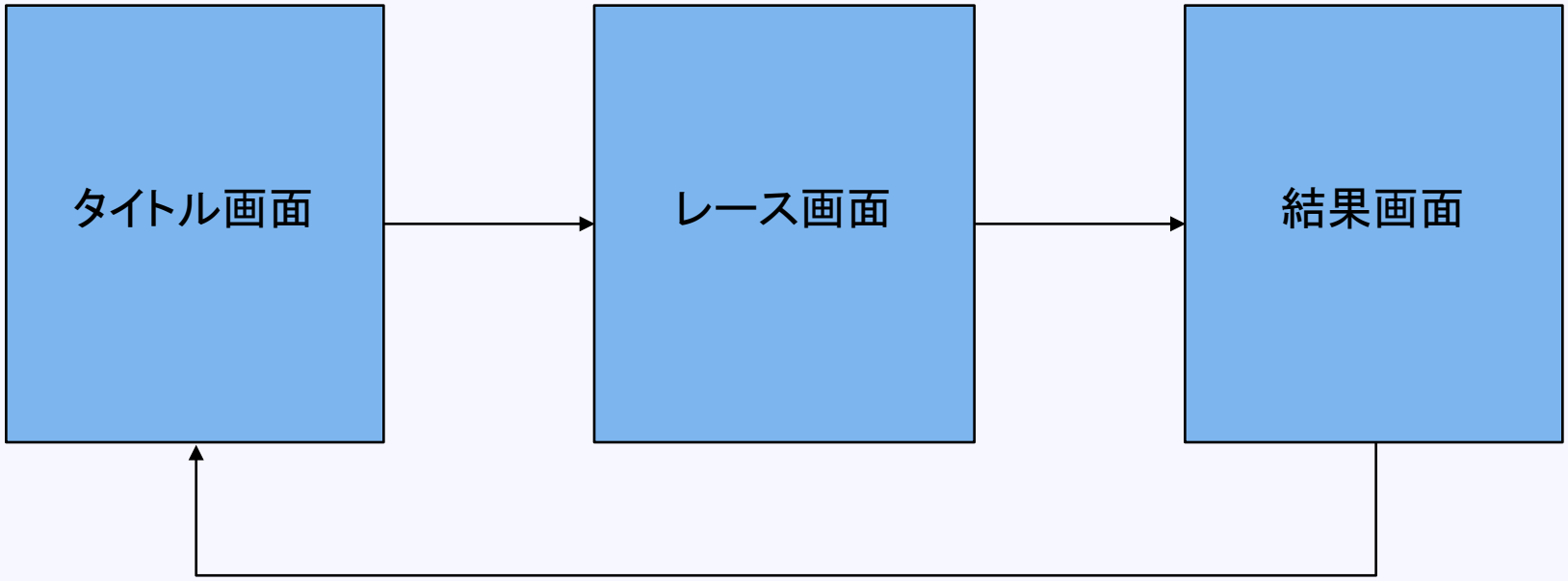
仕様は次のページから。

ポイント！
ルール・仕様を明確にすることで、
開発中の迷いやブレをなくし、効率よく開発することができる。
※企画を考える時にターゲットを明確にするのと同じ

また、必要な素材や実装手順を、実装前に把握できるので、
起こりうる問題にも事前に対処可能。

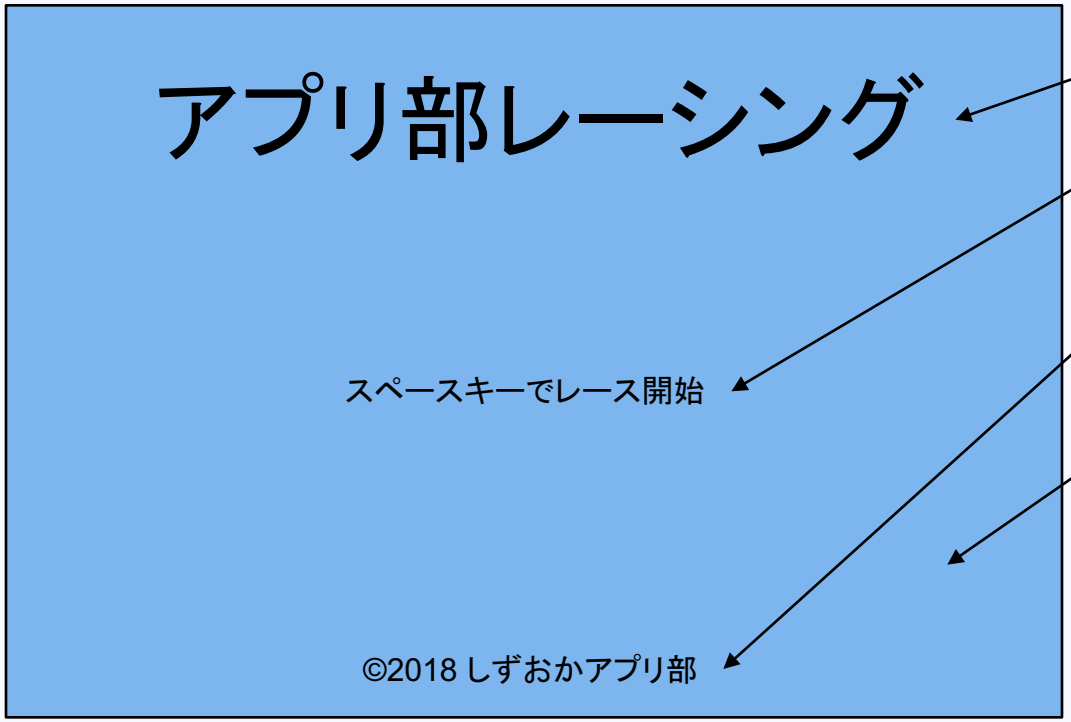
仕様を考える（簡易版）

まずは**画面構成**と**遷移図**



次のページから、簡単に各画面の仕様をまとめていく。

各画面の詳細：タイトル画面



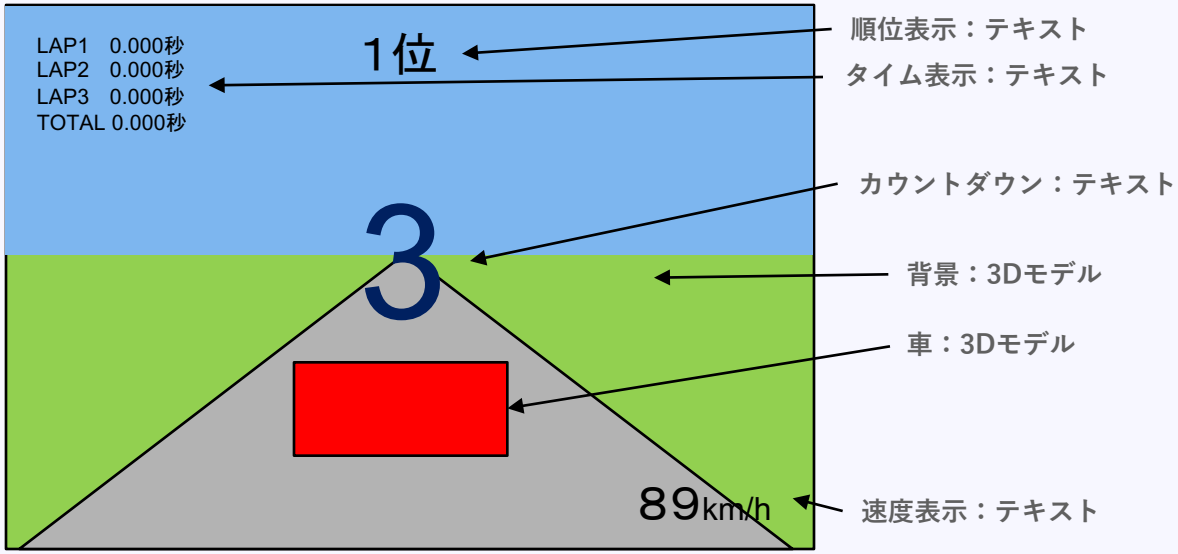
■サウンド
BGM：タイトル画面用
SE：キー押下効果音

■動き
スペースキー押下で、効果音を鳴らし、１秒後にBGMを止め、レース画面へ遷移。

■画面切り替えトランジション
イン・アウトともなし。パッと切り替える。

各画面の詳細：レース画面

カウントダウン時



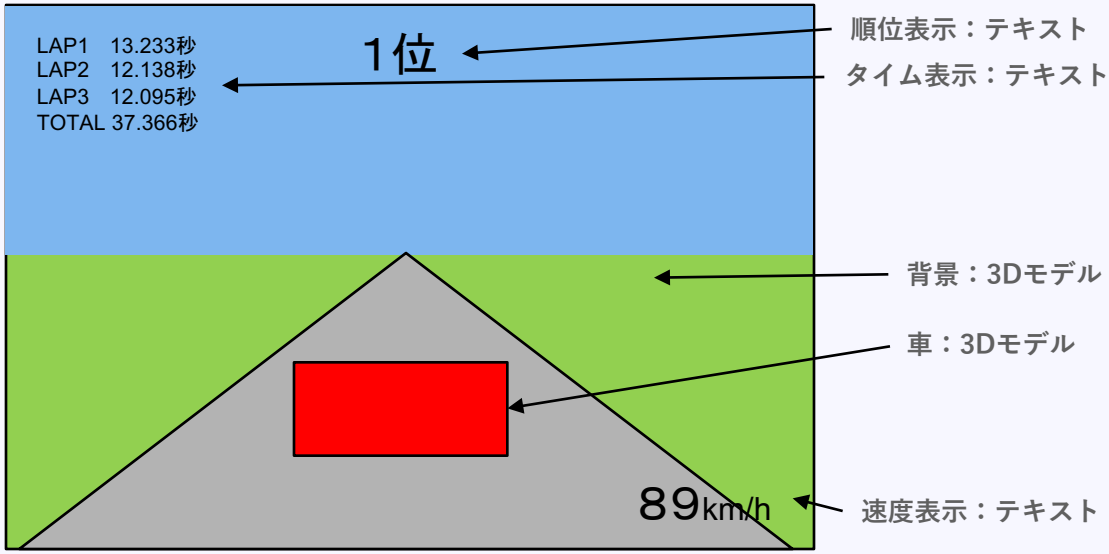
■サウンド
BGM：レース用BGM
SE：カウントダウン用に２種
プップップッピー

■動き
画面遷移１秒後から、３、２、１、GOのカウントダウン。カウントダウンは１秒間隔。
カウントに合わせて効果音を鳴らす。キー入力は無効。
GOになった瞬間、BGM再生開始、キー入力有効に。

■画面切り替えトランジション
インなし。パッと切り替える。

各画面の詳細：レース画面

レース時



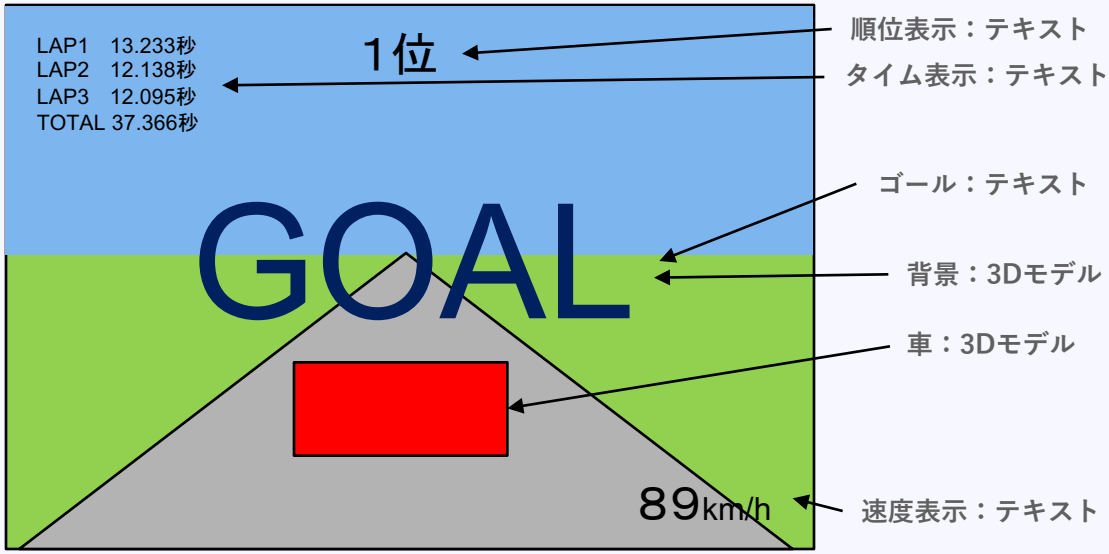
■サウンド
BGM：レース用BGM
SE：エンジン音
ぶつかった時の音
チェックポイント音

■動き
操作方法に従って車を動かす。カメラは車の広報斜め上から見下ろす。
タイム表示、順位表示、速度表示は随時更新。効果音も適宜鳴音。

最高速・旋回性能・敵の強さは、実装しながら調整する。

各画面の詳細：レース画面

ゴール時

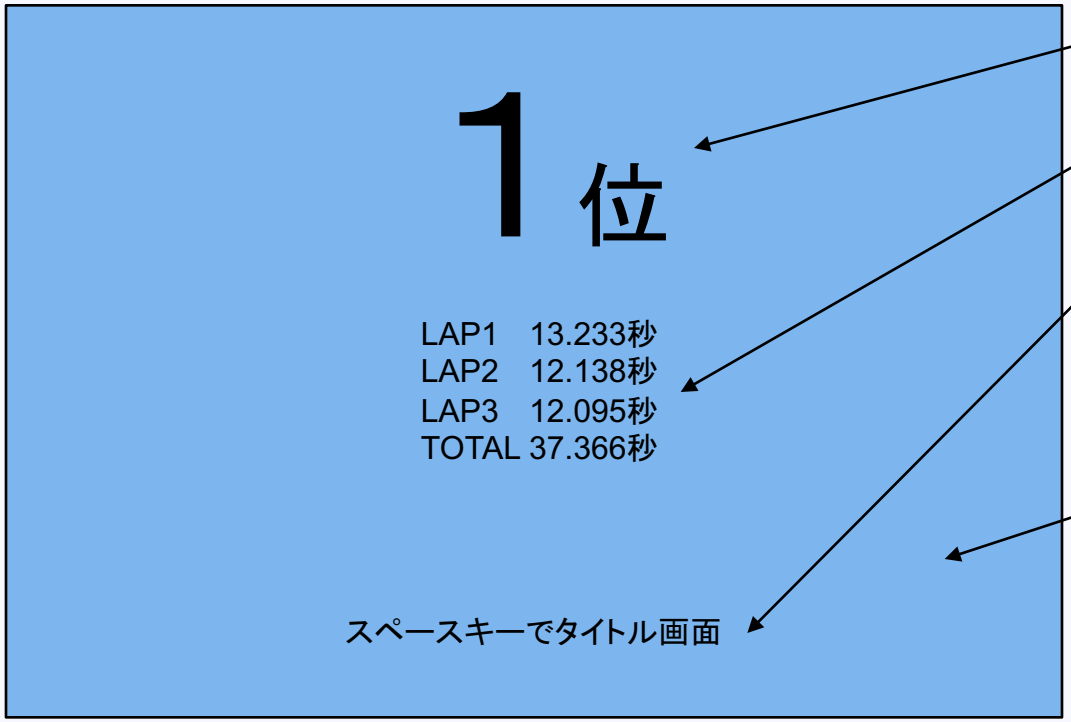


■サウンド
BGM：ゴール用ジングル

■動き
レース用のBGMを止め、ゴール用ジングルを鳴らす。
合わせて画面にGOALテキストを表示し、キー入力を無効。
自車の操作を自動操作に切り替え。
５秒後に結果画面に遷移。

■画面切り替えトランジション
アウトなし。パッと切り替える。

各画面の詳細：結果画面



■サウンド
BGM：結果画面用
SE：キー押下効果音

■動き
スペースキー押下で、効果音を鳴らし、１秒後にBGMを止め、タイトル画面へ遷移。

■画面切り替えトランジション
イン・アウトともなし。パッと切り替える。

必要な素材

画面の仕様を考えると必要な素材がわかる。
簡単にまとめてみる。

画面	種類	内容
タイトル	画像	ロゴ
		背景
	サウンド	タイトルBGM
レース	３Dモデル	コース
		車
	サウンド	レースBGM
		ゴール用ジングル
		エンジン音
		ぶつかった時の音
		チェックポイント
結果	サウンド	結果画面BGM
共通	サウンド	キー押下効果音

レース画面のその他の仕様

画面の仕様は決まったが、車の挙動や周回判定AIなどを
どうやってまとめようか…。

という時は、とりあえず、
必要事項を箇条書きで書き出してみよう。
図で補足できるところは、図でわかりやすくしてみよう。

レース画面のその他の仕様

■車の挙動

・徐々に加速する。
・アクセルを離せば徐々に減速。
・バック動作＝ブレーキとする。
・壁にぶつかったら減速。

・ドリフトできたらいいな。

■操作方法

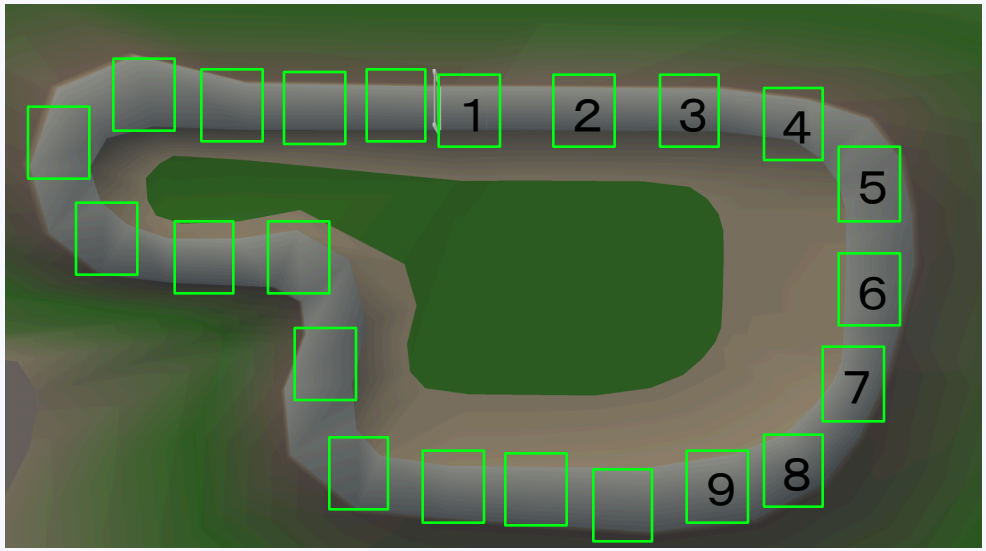
↑キー：前進
↓キー：後退（ブレーキにも）
←キー：左旋回
右キー：右旋回

※調整しやすいつくりとする（インスペクタに表示させる）

レース画面のその他の仕様

■周回チェック、順位チェック方法

・スタート地点から、コースの形状に合わせて複数のチェックポイントを並べておき、順番に通過することで周回チェックを行う。
また、どこのチェックポイントまで通過しているかの比較で順位を決める。
同じチェックポイントにいる場合は、次のチェックポイントに近い方が、順位が上と判定する。

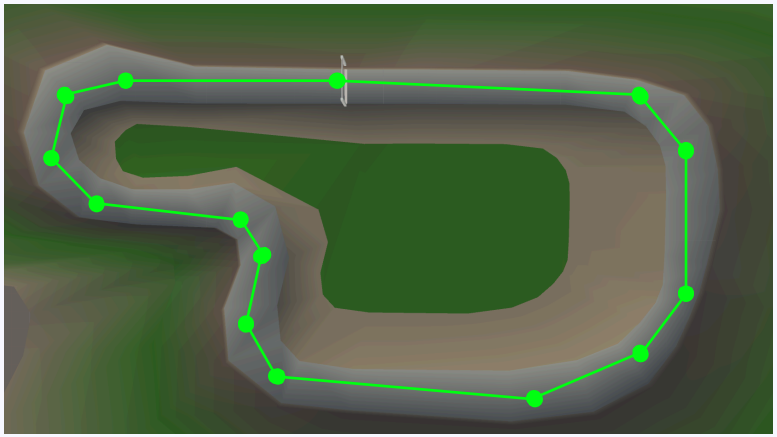


チェックポイント

レース画面のその他の仕様

■AIについて

・プレイヤーと同じように、アクセル、ブレーキ、ハンドル操作を行わせたい。
・周回チェックと同じように、コース上に目的地点を配置して、そこに向かうように操作させる。目的地点にたどり着いたら次の目的地点に変更、を繰り返して、コース上を走るような作りとする。
・上記のチェックポイントを、インコースルート、アウトコースルート、真ん中ルートの３パターンを用意し、リアルタイムに適宜切り替えることで難易度調整を行う。



●目的地点
上図では１ルートだが、３ルート用意して
難易度調整を行えるようにする。

細かいことを言えば仕様書としては全然足りていないが、
指標となるものができた。

次はこの指標を元に設計を行う。

設計

設計の仕方

設計というと難しく感じてしまうので、
最初のうちは、

- **どんな処理が必要なのか**
  その処理をどのスクリプトにさせるか

を考えてみれば良い。

画面ごと考えてみる

■タイトル画面
スクリプト：**TitleManager.cs**
役割：キー判定、シーン切り替え

必要な処理

■結果画面
スクリプト：**ResultManager.cs**
役割：順位表示、タイム表示、
キー判定、シーン切り替え

■レース画面

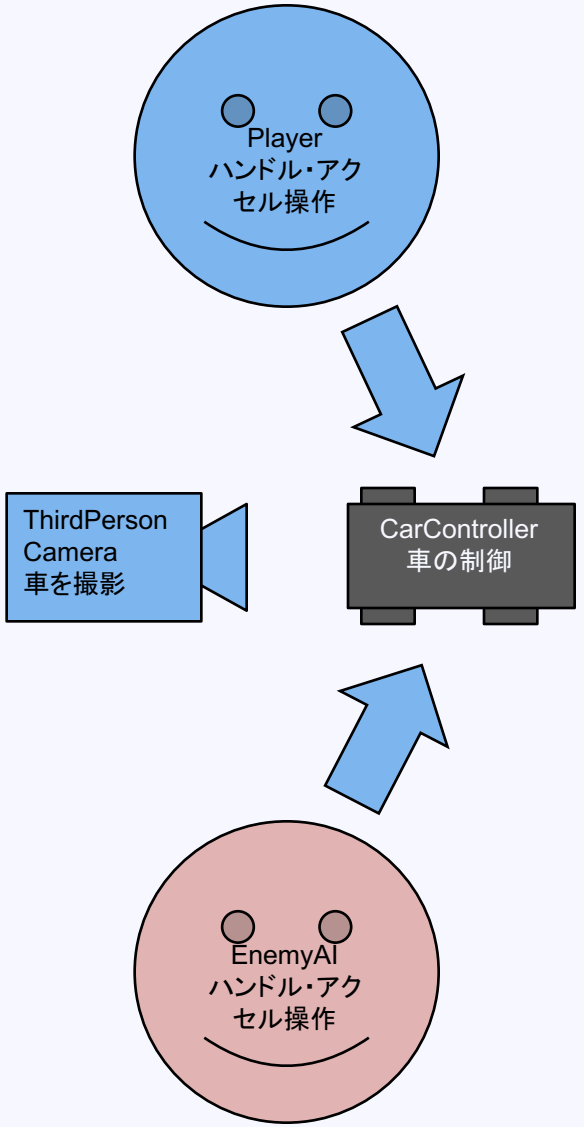
スクリプト：**RaceManager.cs**
役割：カウントダウン、ゴール後処理、
順位表示、タイム表示

スクリプト：**CarController.cs**
役割：車の移動、エンジン音の制御

スクリプト：**Player.cs**
役割：キー入力を受けて車を操作する

スクリプト：**EnemyAI.cs**
役割：CPU車の操作

スクリプト：**ThirdPersonCamera**
役割：車を追うカメラ制御



実装

下準備

作る指標が定まったので、早速実装に。

まずはプロジェクトを滞りなくする進めるために、
整理整頓とルール（命名規則）を明確にする。

整理整頓

フォルダを切る。
あらかじめ必要となるであろうフォルダを作っておく。

- Animations
- Materials
- Models
- Prefabs
- Scenes
- Scripts
- Shaders
- Sounds
- Textures

今回は上記の９つを用意した。どこに何があるか一目でわかる。
規模が大きいゲームは、部品ごとに上記のフォルダ構成を作ると良い。
例えば、Playerフォルダを作り、その中に上記フォルダ群を入れる。
部品単位で完結させるイメージ。そうすることで、他のプロジェクトでも
再利用しやすくなるし、データの受け渡しがしやすくなる。

命名規則

ファイル名の付け方や、
プログラム内での変数名や関数名などの名前の付け方を
統一しておくことで、可読性が上がるメリットがある。

Unityだと、下記の通り。

ファイル名：アッパーキャメルケース
クラス名：アッパーキャメルケース
変数名：ローワーキャメルケース
関数名：アッパーキャメルケース
列挙体：アッパーキャメルケース

アッパーキャメルケース（パスカルケース）
単語の先頭が大文字。 例：DoSomething
ローワーキャメルケース（キャメルケース）
最初の単語は小文字で以降は大文字始まり。例：doSomething

名前は長くなっても良いので、見たら意味がわかる名前をつけることが大切

用語の整理

用語	内容
GameObject	ゲームシーンに出てくるオブジェクトは全てGameObjectという。 （3Dモデル、カメラ、ライトなど）
コンポーネント	GameObjectにくっつける（アタッチ）する部品。 GameObjectに見た目や振る舞いを持たせる部品。 （レンダラ、コライダー、リジッドボディなど） どんなGameObjectにも、位置、向き、大きさを表す**Transformコンポーネント**がアタッチされている。
アセット	素材ファイル。コンポーネントにセットすることで、コンポーネントの見た目や動作を変化させるものもある。 （テクスチャ、マテリアル、オーディオクリップなど）

車の移動について考えてみる

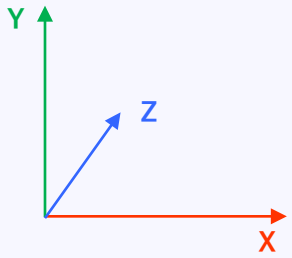
車を移動させるにはどんな方法が良いだろうか？
パッと思いつく方法が、

- **座標を制御する**
- **物理演算で制御する**

の２つ。
リアルな動きを追求するなら物理演算。

座標を制御する方法

Unityの座標の基本は、
X軸が横、Y軸が高さ、Z軸が奥行きとなっている。



なので、X座標とZ座標を制御すれば移動ができる。
ただ、X座標とZ座標を直接いじって車の動きを表現するのはちょっと大変。

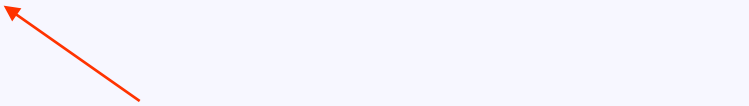
※X座表 = Sin(向き)×移動距離、Z座表 = Cos(向き)×移動距離、でできるが
　これだけだと坂道を登るような状況では移動量が正確でない

なので、Unityの便利な仕組みを使わせてもらう。
日本語で書くと、

移動後の座表 = 現在の座表 +（前方向 × 移動距離）

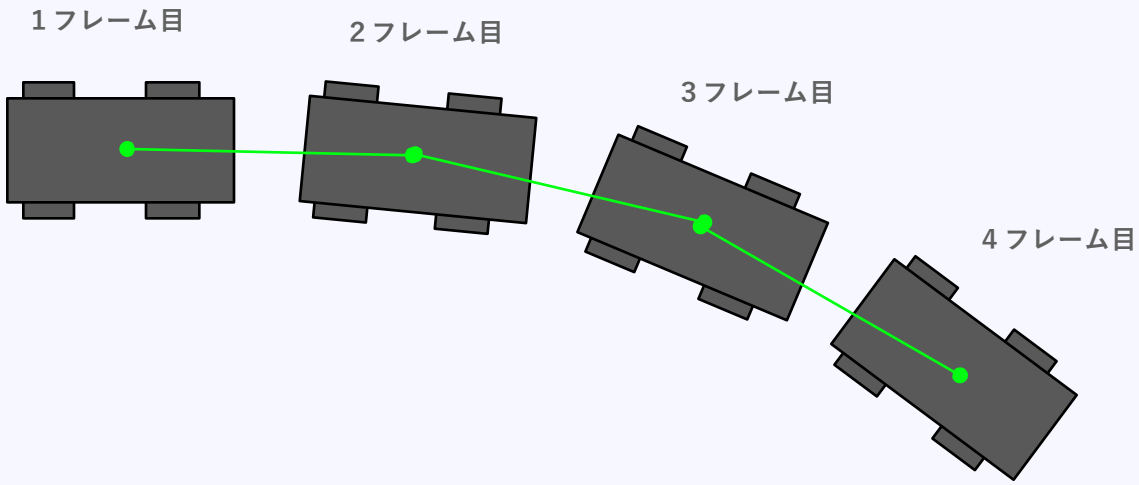
となる。
普通に計算すると**前方向**を求めるのが大変だけど、
Unityでは、**transform.foward**　が前方向を表す。
プログラムで書くと、

```
transform.position += transform.forward * moveDistance
```



という感じ。

考え方としては、
　移動後の座標を毎フレーム求め続ける
ということ。



物理演算で制御する方法

ちゃんと物理演算やるなら、車にタイヤをつけて、
そのタイヤに回転の力を与えて・・・と、
それはそれで大変なので、今回は、車に直接前進する力をかける簡易実装で考える。
物理演算を使う場合は、「移動後の座表」を求めるのではなく、現フレームでどれだけ力を加えるかを考える。

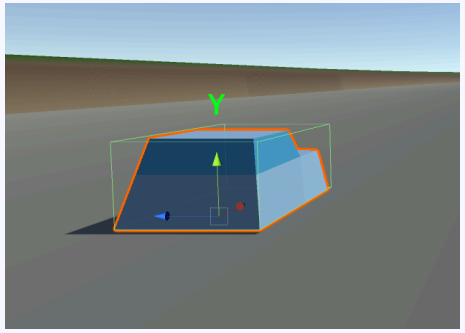
プログラムで書くと

```
rigidBody.AddForce(transform.forward * 力);
```

となる。

車の向き

座標を制御する方法でも物理演算を使う方法でも
車の向きの制御が必要。



向きは、単純にY軸の回転で制御する。

プログラムで書くと、

```
Vector3 rot = transform.rotation.eulerAngles;
rot.y += 変化させたい回転量;
transform.rotation = Quaternion.Euler(rot);
```

Transformコンポーネントから回転情報を受け取る

受け取った回転情報を元に回転値に変化をつける

変更した回転値をTransformコンポーネントに反映

※回転させる関数もあるが、仕組みを知ってもらうために、
上記のように記述。

１フレームあたりの移動量

ゲームは１秒間に何十コマもの絵を切り替えて、
なめらかにアニメーションしているように見せている。

コマのことを**フレーム**という。
１秒間のコマ数のことを**フレームレート**という。

フレーム数が多い方がなめらかに見えるが、高画質なグラフィックにすればするほど処理に時間がかかり、フレームレートが落ちてしまう。また、パソコンやスマホの性能にも影響される。

ここでひとつ考えなくてはいけないことがある。

１フレームに２m進むというような実装をしてしまうと、
フレームレートが違う環境では、1秒後に進む距離が
変わってしまう問題が発生する。

なので、考え方を「１秒に何m進むか」とする必要がある。
この考え方で１フレームの移動量を求める計算式は、

１フレームの移動量＝移動距離×１フレームの処理時間

となる。
Unityでは、１フレームの処理時間は、
Time.deltaTime
で表される。

というところで続きは次回。
次回は実装中心となります。

ご清聴ありがとうございました